

AI PIMPLE CARE

1.PROBLEM STATEMENT

The current skincare industry lacks a personalized and efficient solution for individuals struggling with facial pimples and selecting appropriate cosmetic products tailored to their unique skin conditions. Many consumers face challenges in identifying effective products, resulting in suboptimal skincare routines and dissatisfaction with their outcomes. This project aims to address these challenges by developing an AIbased application, named AISkincare, that detects facial pimples through advanced image processing and provides personalized cosmetic product recommendations based on individual skin deficiencies. The key problems to be tackled include Inaccurate Pimple Detection, Limited Personalization in Product Recommendations, UserFriendly Interface, Validation and Trustworthiness, Privacy concerns. This project seeks to fill the gap in the skincare industry by providing a comprehensive, accurate, and userfriendly solution for pimple detection and cosmetic product recommendations. AISkincare aims to empower users with personalized insights, facilitating improved skincare routines and promoting overall skin health.

2.SOFTWARE REQUIREMENT SPECIFICATION

2.1 INTRODUCTION

2.1.1 PURPOSE

The purpose of this document is to specify the requirements for the development of a comprehensive AIbased application that uses facial recognition technology to detect pimples on the face and recommends personalized cosmetic products based on the identified deficiencies. The application seeks to go beyond basic pimple detection, aiming to deliver personalized recommendations for cosmetic products. By analyzing identified deficiencies in the user's skin, the system will offer tailored suggestions, empowering users to address specific skincare concerns with targeted solutions. The ultimate goal is to empower users with the knowledge to make informed skincare choices. By combining advanced technology with personalized insights, the application aspires to be a valuable resource in the user's skincare journey, promoting healthier and more confident skin.

2.1.2 SCOPE

The application will cater to individuals seeking a personalized skincare solution. It will analyze facial images, identify pimples, assess overall skin conditions, and provide recommendations for cosmetic products. The system will focus on userfriendly interaction and realtime analysisIn addition to end users, the scope includes skincare professionals such as dermatologists, estheticians, and skincare consultants. These professionals can leverage the system to enhance their services, utilizing the detailed analysis results for more accurate skincare assessments. The application's scope extends to individuals of all ages and genders who are proactively seeking personalized skincare solutions. By catering to a diverse audience, the system aims to address a wide range of skincare needs and preferences.

2.1.3 REFERENCE

GitHub Repositories:

Explore public GitHub repositories for similar projects. While you won't copy their project, you can get ideas on how others structure their requirements and the level of detail they include.

Online project Templates:

Several websites offer project templates that you can use as a starting point. However, be sure to customize them according to your project's specific needs.

2.2 OVERALL DESCRIPTION

2.2.1 PRODUCT PERSPECTIVE

The AIPimple care will be a standalone android mobile application that interacts with firebase for pimple samples data and integrates with relevant APIs for assistance program information. The goal is to allow users greater and easier access to the skin care routine.

2.2.2 PRODUCT FEATURES

We can subdivided the project into main features. Details of each of the following functions can be found in the following.

I) Register

The registration process allows unregistered users to enroll and create a new account with the application. Users will be required to fill out a registration form, providing essential information such as first name, last name, email address, and password. The system will perform validation checks to ensure that all required data is provided. If necessary, the system may prompt users to enter additional information to enhance their profile, such as skincare preferences or specific skin concerns.

II) Login

Registered users can log in to their accounts using their email and password.

The system will verify the entered email and password to authenticate the user.

If a user is not registered, the application will provide an option to redirect the user to the registration process.

III) Face Detection

This feature employs an AI model trained to detect and identify pimples on the user's face.

Users will capture facial images using the device's camera.

The AI model will analyze these images, identifying the presence, type, and severity of pimples.

The results of the face detection process will be visually displayed on the user's screen.

Users will see a comprehensive visual representation, such as a pimple map, highlighting the location and severity of identified pimples.

Clear and userfriendly visuals will be presented to enhance user understanding.

IV) Product Recommendations

The system will analyze the identified pimples and assess overall skin conditions to identify deficiencies.

Deficiencies may include issues such as dehydration, excess oil production, or specific skincare needs.

The analysis will be conducted based on AI algorithms trained to recognize patterns in the user's skin health. The system will utilize advanced algorithms to match identified deficiencies with suitable cosmetic products.

A database of cosmetic products will be integrated, with products categorized based on their effectiveness in addressing specific skin concerns.

Recommendations will be personalized, considering the severity and type of identified pimples and overall skin deficiencies.

V) Appointment

Users can book appointments with skincare professionals, including dermatologists and estheticians.

A userfriendly booking system will be implemented, allowing users to select preferred dates and times for consultations.

Notifications and reminders will be sent to users to ensure they don't miss their appointments.

Users can set reminders for their skincare routines and followup appointments.

The system will feature a reminder functionality, allowing users to schedule and manage skincare routines.

Reminders can be customized based on user preferences and the recommended skincare regimen.

2.3 USER CLASS and CHARACTERISTICS

USER CLASS:

End users are individuals who utilize the Pimple Detection and Cosmetic Recommendation System for personal skincare needs. They may have varying levels of knowledge about skincare products and desire a personalized solution for pimple detection and product recommendations. Skincare professionals include dermatologists, estheticians, and skincare consultants who leverage the application to enhance their services. They may use the system as a supplementary tool in their practice to validate and augment their skincare recommendations.

USER CHARACTERISTICS:

- Diverse User Base
- Skincare Goals
- Interaction Preferences
- Geographical Diversity
- Appointment Management
- Feedback and Improvement
- Accessibility Considerations
- Desire for Informed Choices
- Internet Connectivity
- Commitment to Skincare

Users committed to following skincare routines and recommendations for improved skin health. End users may include individuals of different ages, genders, and ethnicities, each with unique skincare needs. Varied levels of technical proficiency, from techsavvy individuals to those less familiar with mobile applications. Skincare professionals possess in-depth knowledge of skincare, allowing them to interpret and use the detailed analysis results provided by the application. Professionals may integrate the system into their existing skincare practice, relying on its insights for personalized client recommendations. Capture clear and accurate facial images for analysis. Provide feedback on the effectiveness of recommended skincare products. Follow recommended skincare routines and product usage guidelines.

2.4 EXTERNAL INTERFACE REQUIREMENTS

2.4.1 User Interfaces:

The primary interface through which end users and skincare professionals interact with the system.

The interface should be intuitive, userfriendly, and responsive to touch gestures.

Support for multiple languages to accommodate a diverse user base.

Clear instructions for capturing facial images and navigating through analysis results and product recommendations.

Inapp tooltips and guidance to assist users in understanding analysis results.

2.4.3Communication Interfaces

The application requires a stable internet connection for realtime analysis and updates.

Graceful handling of intermittent connectivity, allowing users to capture images offline and sync data when connectivity is restored.

Efficient data transfer to minimize the impact on user experience.

support through email or an integrated chat system.

2.5 OTHER NONFUNCTIONAL REQUIREMENTS

2.5.1 Performance Requirements

The system should provide quick responses to user actions for a seamless user experience. Pimple detection and analysis should occur within a maximum of 5 seconds. Cosmetic product recommendations should be generated within 10 seconds of completing the analysis. The system should scale to accommodate a growing user base. The application should handle a minimum of 100 simultaneous users without significant performance degradation. Scalability testing should be conducted periodically to ensure performance as the user base expands.

2.5.2 Security Requirements

Protecting user data, especially facial images and skincare information, is critical. Facial images and skincare data should be securely stored and transmitted using encryption (HTTPS). Compliance with data protection regulations and standards, ensuring user privacy. Ensuring that only authorized users can access sensitive information. Secure user authentication mechanisms, such as password protection or biometric authentication. Skincare professionals accessing the webbased dashboard should have additional authentication layers.

2.5.3 Software Quality Attributes:

Usability

The system should be intuitive and userfriendly for individuals with varying technical proficiency. A user interface that provides clear instructions for capturing facial images and understanding analysis results. Support for touch gestures and other common user interaction patterns.

Reliability

The system should consistently deliver accurate results and recommendations. Pimple detection and skin analysis algorithms should have a minimum accuracy rate of 95%. Regular system maintenance to ensure uninterrupted service.

Maintainability

The system should be easily maintainable to incorporate updates and improvements. Well documented codebase and system architecture for ease of understanding. Version control and continuous integration practices for efficient development and updates.

Accessibility

The system should be accessible to users with varying abilities and disabilities. Compliance with accessibility standards (e.g., WCAG) to ensure a userfriendly experience for individuals with visual or auditory impairments.

3.UNIFIED MODELING LANGUAGE(UML) DIAGRAMS

3.1 USE CASE DIAGRAM

A Use Case Diagram provides a highlevel view of the functionalities and interactions within a system from the perspective of its users. For the AI Pimple Detection App, the use case diagram outlines the various actions or use cases that users (actors) can perform and how these interactions take place. Let's break down the key elements of the use case diagram:

1. Actors:

User:

Represents the end user of the AI Pimple Detection App.

Engages in various interactions with the application.

Skincare Professional:

Represents skincare professionals, such as dermatologists or estheticians.

Utilizes the system for professional skincare assessments.

2. Use Cases:

Register:

The user can register a new account, providing necessary information.

Login:

Registered users can log in to their accounts using their credentials.

Capture Facial Image:

Users can capture facial images using the device's camera.

Face Detection:

The system uses AIbased facial recognition to detect and identify pimples on the user's face.

Display Pimple Analysis:

The results of the face detection process, including a visual representation of identified pimples, are displayed to the user.

Receive Product Recommendations:

Users receive personalized cosmetic product recommendations based on the analysis results.

Book Appointment:

Users, including skincare professionals, can book appointments for consultations.

Set Skincare Reminders:

Users can set reminders for skincare routines and followup appointments.

Provide Feedback:

Users can provide feedback on recommended products, contributing to system improvement.

3. Associations:**User and Skincare Professional:**

Both are associated with various use cases, indicating their interaction with the system.

User and AI Pimple Detection System:

Show the primary interactions between the user and the system.

Skincare Professional and AI Pimple Detection System:

Indicate how skincare professionals interact with the system.

4. System Boundary:**AI Pimple Detection App:**

Represents the overall system, encapsulating all the described use cases and interactions.

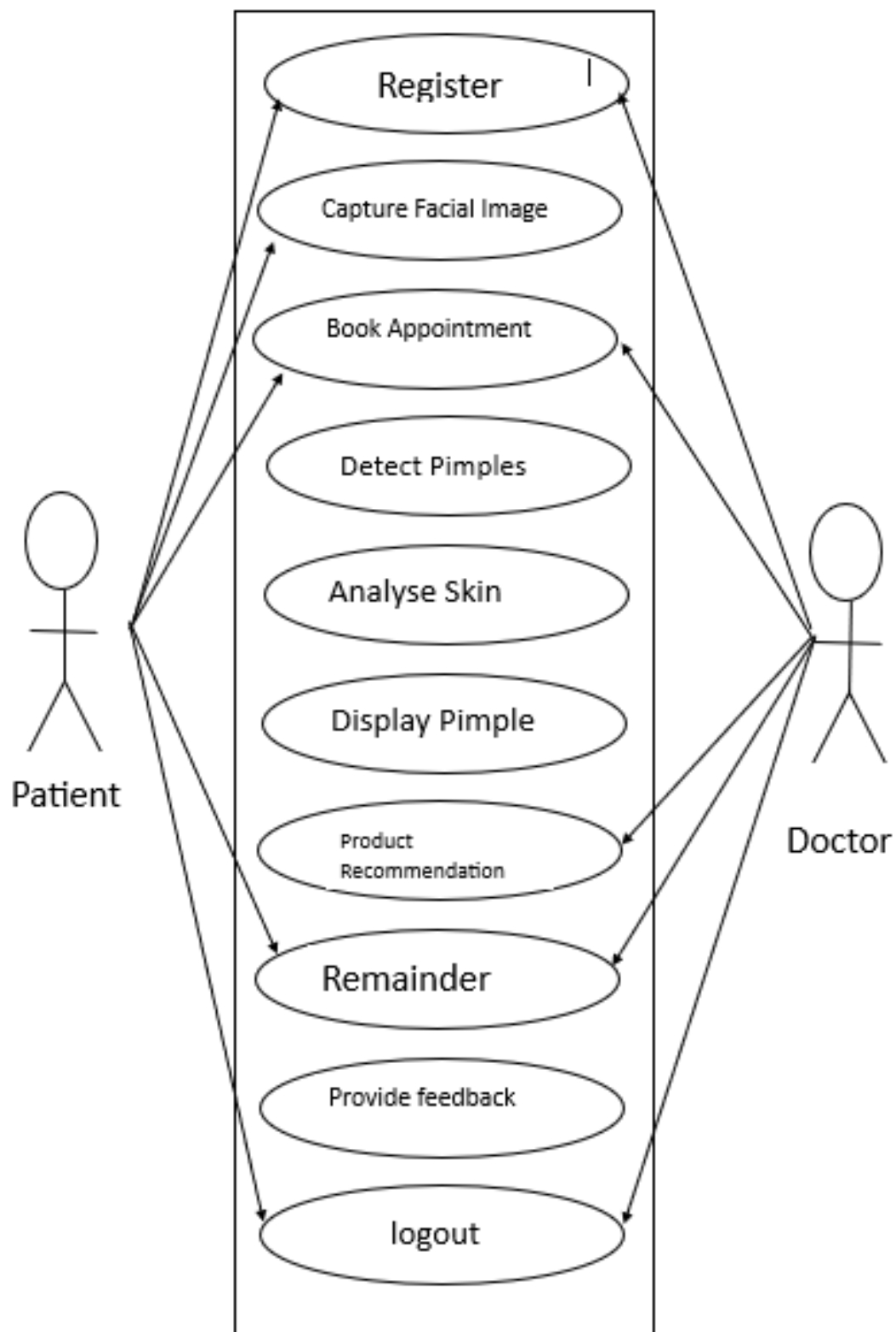
5. Extend Relationships:**Extended by Feedback and Improvement:**

Signifies that the "Provide Feedback" use case contributes to continuous improvement.

Extended by Face Detection and Display Pimple Analysis:

Indicates that face detection is a prerequisite for displaying pimple analysis results.

USE-CASE DIAGRAM:



3.2 CLASS DIAGRAM

A Class Diagram provides a structural view of a system by illustrating the classes, their attributes, methods, and the relationships between them. In the context of the AI Pimple Detection App, the class diagram represents the key entities and their interactions within the system. Here's an explanation of the main classes and their relationships:

1. User Class:

Represents the end users of the AI Pimple Detection App.

Attributes: `userId`: Unique identifier for each user.

`firstName`, `lastName`: User's name for personalization.

`email`: User's email address for authentication.

`password`: Securely stored password for account access.

Methods: `register()`: Method for user registration.

`login()`: Method for user login.

`captureFacialImage()`: Method to capture facial images.

`setSkincareReminder()`: Method to set skincare reminders.

2. SkincareProfessional Class:

Represents skincare professionals who may use the app for professional assessments.

Attributes: `professionalId`: Unique identifier for each skincare professional.

`fullName`: Full name of the skincare professional.

`specialization`: Area of expertise (e.g., dermatology, esthetics).

Methods: `bookAppointment()`: Method for booking appointments.

`analyzeSkinCondition()`: Method for professional skincare analysis.

3. AI Pimple Detection System Class:

Represents the core functionality of the app related to pimple detection and analysis.

Attributes: `analysisId`: Unique identifier for each analysis session.

`resultData`: Data structure to store the results of facial analysis.

Methods: `detectPimples()`: Method to perform AIbased pimple detection.
 `generateProductRecommendations()`: Method to recommend cosmetic products based on analysis results.

4. Appointment Class:

Represents user appointments for consultations with skincare professionals.

Attributes: `appointmentId`: Unique identifier for each appointment.
 `dateTime`: Date and time of the appointment.
 `user`: Reference to the User class.
 `skincareProfessional`: Reference to the SkincareProfessional class.

Methods: `scheduleAppointment()`: Method for scheduling appointments.

5. ProductRecommendation Class:

Represents recommended cosmetic products based on user analysis.

Attributes: `productId`: Unique identifier for each recommended product.
 `productName`: Name of the recommended cosmetic product.
 `deficiencyType`: Type of skin deficiency the product addresses.
 `user`: Reference to the User class.
Methods: `displayRecommendations()`: Method to display recommended products to the user.

6. Feedback Class:

Represents user feedback on recommended products for system improvement.

Attributes: `feedbackId`: Unique identifier for each feedback entry.
 `user`: Reference to the User class.
 `productRecommendation`: Reference to the ProductRecommendation class.
 `comment`: User's feedback comment.
Methods: `submitFeedback()`: Method for users to submit feedback.

7. SkincareReminder Class:

Represents userset reminders for skincare routines and followup appointments.

Attributes:

`reminderId`: Unique identifier for each reminder.

`user`: Reference to the User class.

`dateTime`: Date and time of the reminder.

`reminderType`: Type of reminder (e.g., skincare routine, followup appointment).

Methods:

`setReminder()`: Method for users to set reminders.

8. Relationships:

Association:

User and Appointment: OnetoMany relationship indicating that a user can have multiple appointments.

User and ProductRecommendation: OnetoMany relationship indicating that a user can receive multiple product recommendations.

User and Feedback: OnetoMany relationship indicating that a user can provide multiple feedback entries.

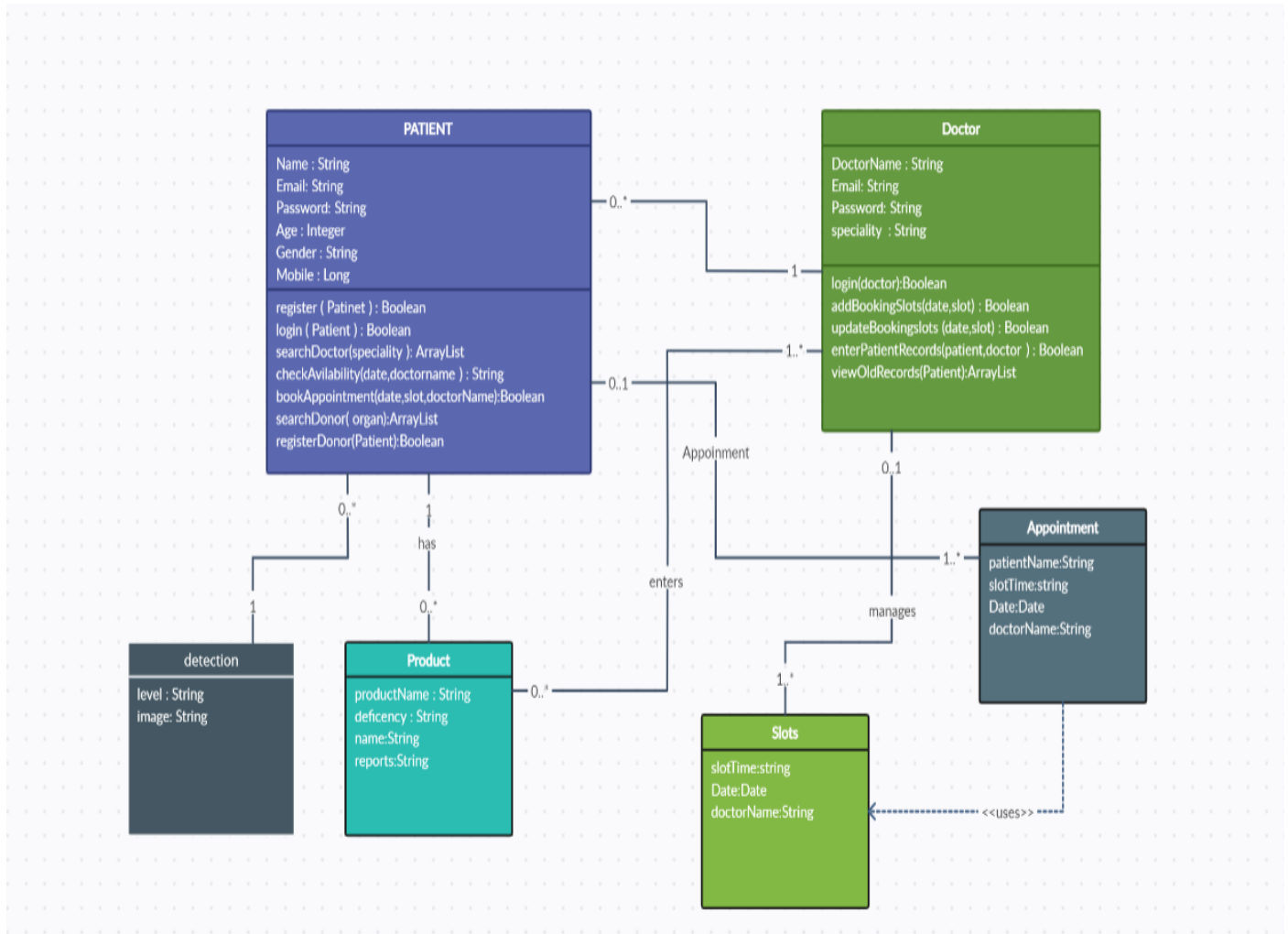
User and SkincareReminder: OnetoMany relationship indicating that a user can set multiple reminders.

SkincareProfessional and Appointment: OnetoMany relationship indicating that a skincare professional can have multiple appointments.

Inheritance:

SkincareProfessional and User: Inheritance relationship indicating that a SkincareProfessional is a type of User.

CLASS DIAGRAM:



3.3 INTERACTION DIAGRAM

It is clear that the diagram is used to describe some type of interactions among the different elements in the model. This interaction is a part of dynamic behavior of the system. This interactive behavior is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram. The basic purpose of both the diagrams are similar.

Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

The purpose of interaction diagram is –

To capture the dynamic behaviour of a system.

- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

3.3.1 SEQUENCE DIAGRAM

This Sequence Diagram outlines the dynamic interactions between the User, the AI Pimple Detection System, and the User Interface. It provides a stepbystep visualization of how the system processes the user's facial image, detects pimples, and recommends cosmetic products. Each step represents a message or action between different components of the system, illustrating the flow of information and the sequence of events during this specific user interaction.

Actors:

User

AI Pimple Detection System

User Interface (UI)

Steps:

User Initiates Image Capture:

The user triggers the image capture process through the app interface.

UI Sends Capture Request to AI System:

The User Interface sends a message to the AI Pimple Detection System to initiate the capture process.

AI System Captures Facial Image:

The AI Pimple Detection System captures the user's facial image using the device's camera.

AI System Processes Image for Pimple Detection:

The AI Pimple Detection System processes the captured image to detect and identify pimples.

AI System Sends Pimple Analysis Results to UI:

The AI Pimple Detection System sends the analysis results (identified pimples) back to the User Interface.

UI Displays Pimple Analysis Results to User:

The User Interface displays a visual representation of identified pimples on the user's face.

UI Requests Product Recommendations from AI System:

The User Interface sends a request to the AI Pimple Detection System for personalized cosmetic product recommendations based on the analysis.

AI System Generates Product Recommendations:

The AI Pimple Detection System utilizes algorithms to generate personalized cosmetic product recommendations.

AI System Sends Product Recommendations to UI:

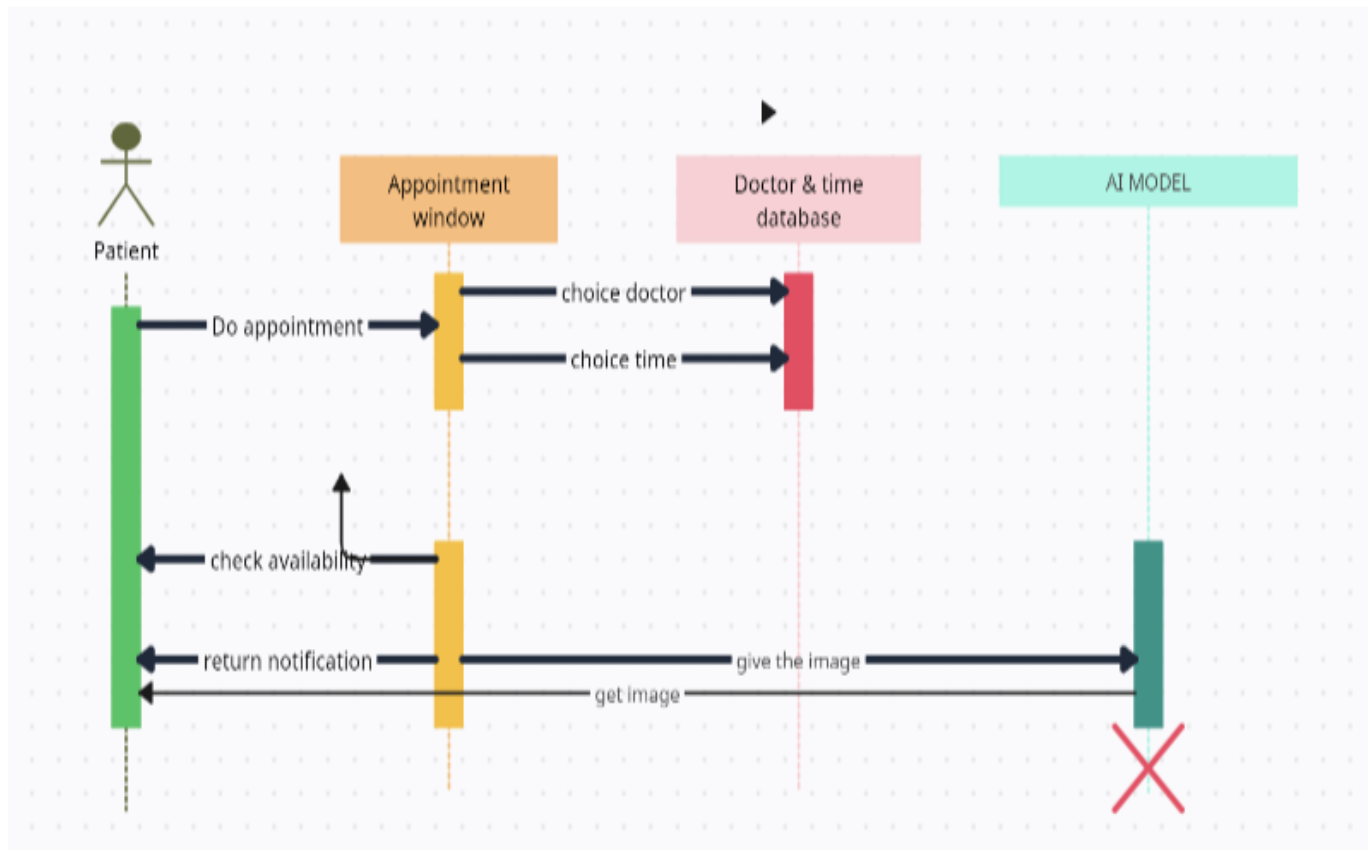
The AI Pimple Detection System sends the generated product recommendations back to the User Interface.

UI Displays Product Recommendations to User:

The User Interface presents the recommended cosmetic products to the user.

End of Interaction:

The sequence concludes with the user having received pimple analysis results and personalized cosmetic product recommendations.



3.3.2 COLLABORATION DIAGRAM:

A Collaboration Diagram, also known as a Communication Diagram, visualizes the interactions and relationships between objects or components in a system. In the context of the AI Pimple Detection and Cosmetic Recommendation System, the Collaboration Diagram provides a dynamic view of how various system components collaborate to achieve specific functionalities.

Components involved are:

- 1. User:** Represents the end user interacting with the system.
- 2. User Interface (UI):** Represents the graphical user interface through which the user interacts.
- 3. AI Pimple Detection System:** Represents the core system responsible for pimple detection and product recommendations.
- 4. Product Recommendation Service:** A service within the system responsible for generating cosmetic product recommendations.

Interactions:

1. User Initiates Image Capture:

The User sends a message to the User Interface to initiate the image capture process.

2. UI Requests Image Capture from AI System:

The User Interface sends a message to the AI Pimple Detection System, requesting the capture of a facial image.

3. AI System Captures Facial Image:

The AI Pimple Detection System captures the facial image and sends a confirmation message to the User Interface.

4. AI System Processes Image for Pimple Detection:

The AI Pimple Detection System processes the captured image for pimple detection and sends a message to the Product Recommendation Service to begin analysis.

5. Product Recommendation Service Analyzes Image:

The Product Recommendation Service analyzes the facial image and identifies pimples, sending the results back to the AI Pimple Detection System.

6. AI System Sends Pimple Analysis Results to UI:

The AI Pimple Detection System sends the pimple analysis results to the User Interface for display.

7. UI Displays Pimple Analysis Results to User:

The User Interface displays a visual representation of identified pimples on the user's face.

8. UI Requests Product Recommendations from AI System:

The User Interface sends a message to the AI Pimple Detection System, requesting personalized cosmetic product recommendations based on the analysis.

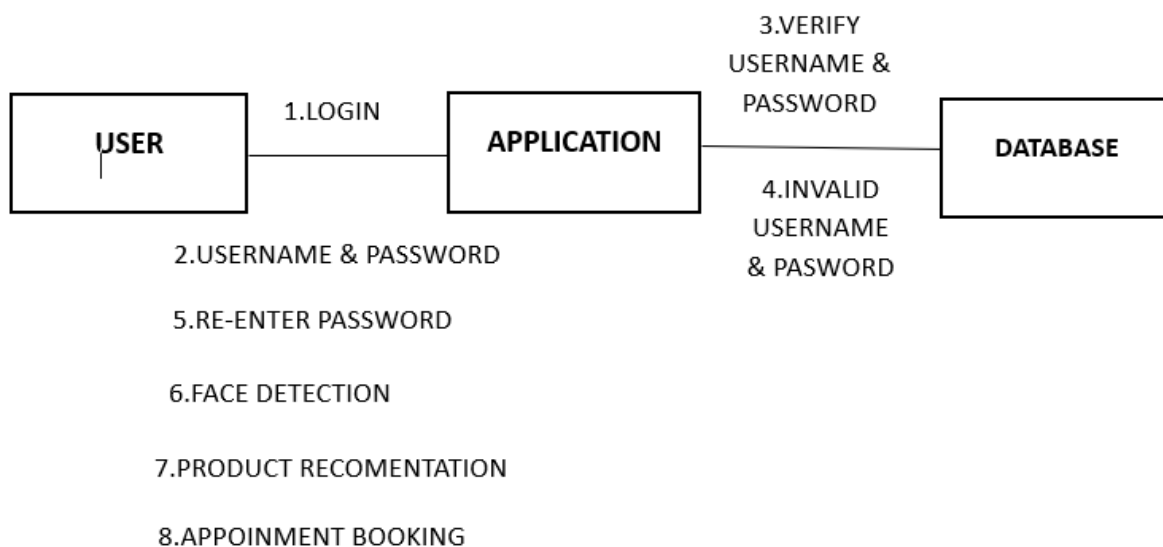
9. AI System Generates Product Recommendations:

The AI Pimple Detection System utilizes algorithms to generate personalized cosmetic product recommendations and sends the results to the User Interface.

10. UI Displays Product Recommendations to User:

The User Interface presents the recommended cosmetic products to the user.

COLLABORATION DIAGRAM:



3.4 STATECHART DIAGRAM AND ACTIVITY DIAGRAM

Statechart Diagram:

States:

1. Idle:

Initial state when the user is not actively using the app.

2. Capturing Image:

The system transitions to this state when the user initiates the image capture process.

3. Processing Image:

The state when the AI system is actively processing the captured facial image for pimple detection.

4. Displaying Results:

Transitioned to when the system is displaying the results of pimple detection to the user.

5. Recommending Products:

The state in which the system generates and displays personalized cosmetic product recommendations.

Transitions:

Idle to Capturing Image:

Triggered when the user initiates the image capture process.

Capturing Image to Processing Image:

Triggered when the system captures the facial image and begins processing for pimple detection.

Processing Image to Displaying Results:

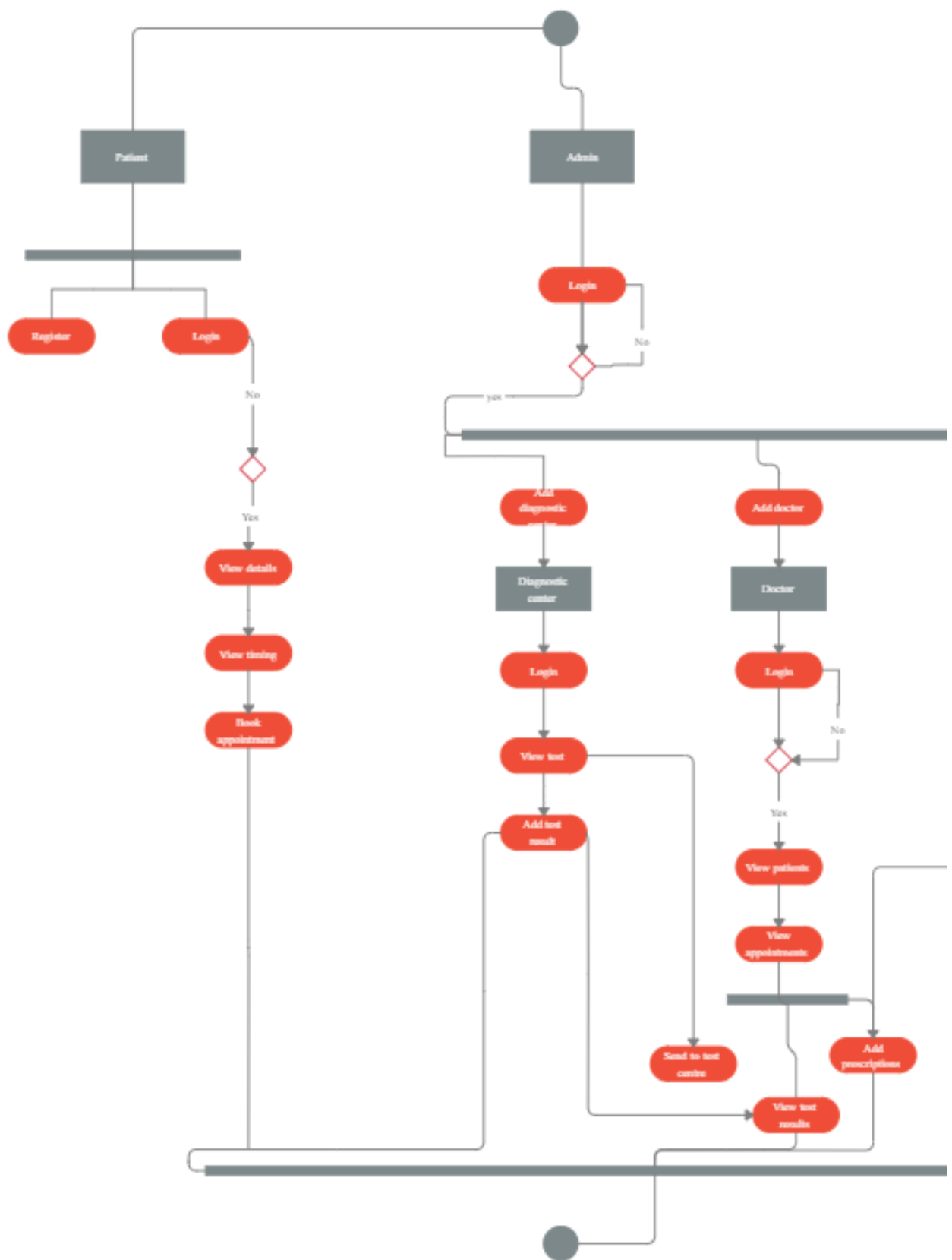
Transitioned when the system completes the pimple detection process and is ready to display the results.

Displaying Results to Recommending Products:

Triggered when the user requests personalized cosmetic product recommendations based on the pimple analysis.

Recommending Products to Idle:

Transitioned when the user completes the interaction, returning to the idle state.



Activity Diagram:

Activities:

1. Capture Facial Image:

The process of capturing a facial image initiated by the user.

2. Process Image for Pimple Detection:

The AI system processes the captured image for pimple detection.

3. Display Pimple Analysis Results:

Displaying the results of the pimple analysis to the user.

4. Request Product Recommendations:

The user requests personalized cosmetic product recommendations based on the pimple analysis.

5. Generate Product Recommendations:

The AI system generates personalized cosmetic product recommendations.

6. Display Product Recommendations:

Displaying the recommended cosmetic products to the user.

Control Flow:

Capture Facial Image:

Initiated by the user, leading to the processing of the captured image.

Process Image for Pimple Detection:

The AI system processes the image for pimple detection, leading to the display of results.

Display Pimple Analysis Results:

The system displays the results, and the user decides whether to request product recommendations.

Request Product Recommendations:

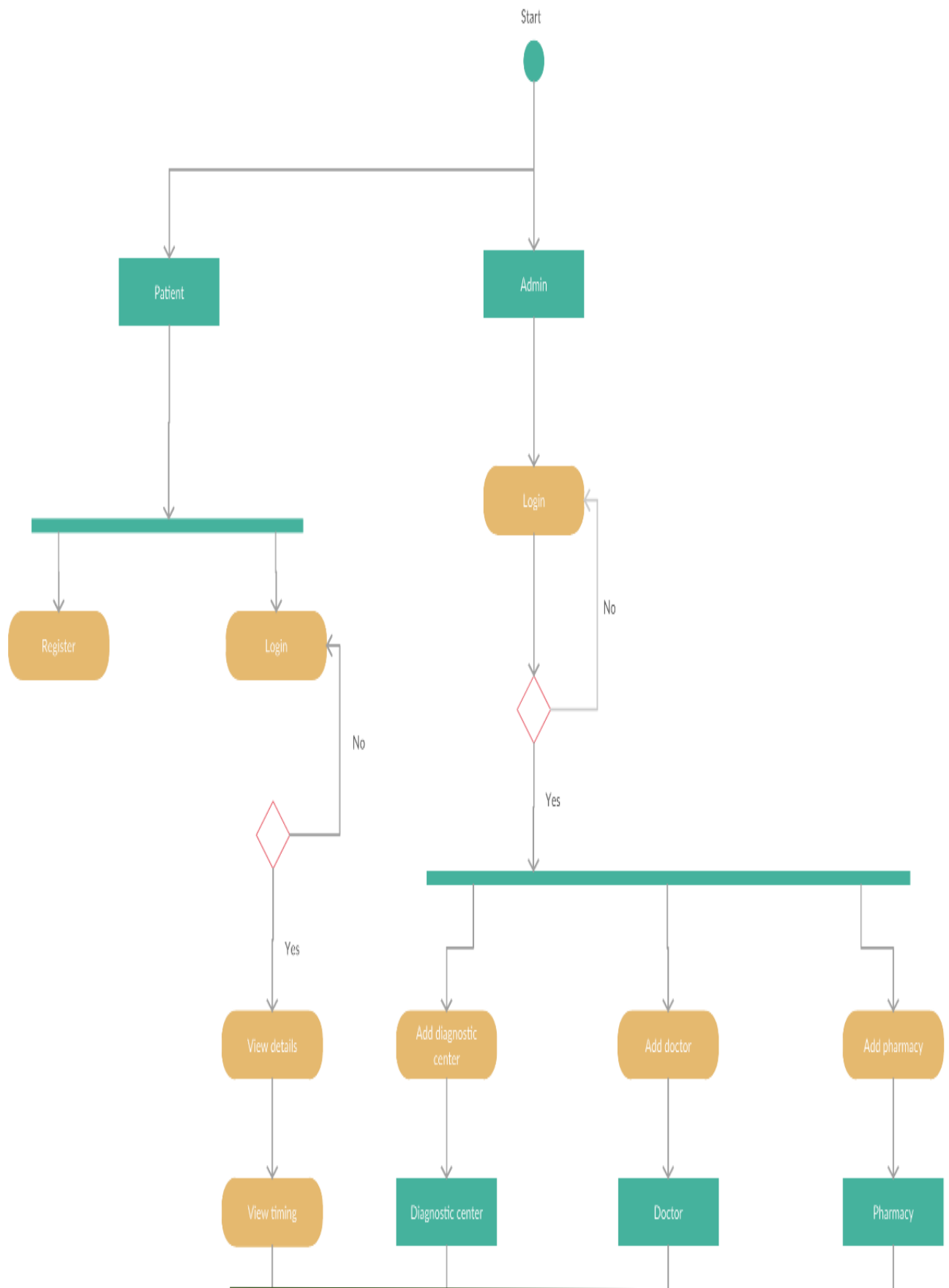
If the user requests recommendations, the system proceeds to generate and display personalized cosmetic product recommendations.

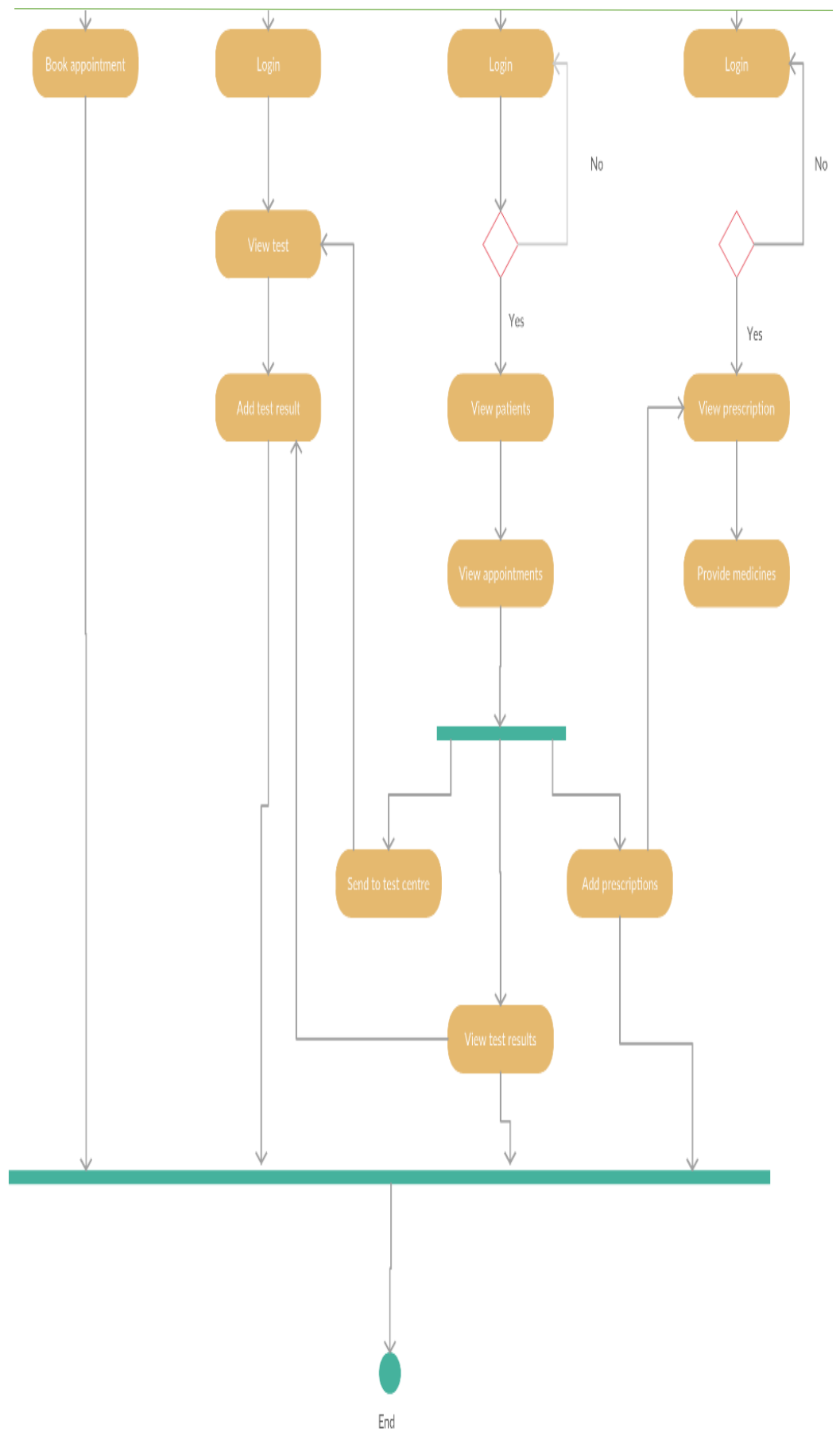
Generate Product Recommendations:

The AI system generates product recommendations based on the pimple analysis.

Display Product Recommendations:

The recommended products are displayed to the user, completing the interaction.





3.5 PACKAGE DIAGRAM

A Package Diagram is a type of diagram in the Unified Modeling Language (UML) that organizes and shows the dependencies between different packages within a system. Packages are used to group related elements, such as classes, interfaces, and other packages, to provide a modular and organized structure. In the context of the AI Pimple Detection and Cosmetic Recommendation System, let's create a hypothetical Package Diagram.

Package Diagram:

Packages:

1. User Interface (UI):

Contains classes and components related to the user interface, including screens, buttons, and user interaction logic.

2. AI Pimple Detection System:

Encompasses the core functionality of pimple detection and product recommendations.

Subpackages:

Image Processing:

Contains classes and components related to capturing and processing facial images.

Pimple Detection:

Includes classes for the AI model and algorithms responsible for pimple detection.

Product Recommendations:

Consists of classes for generating and presenting personalized cosmetic product recommendations.

3. Skincare Professionals:

Packages related to features specific to skincare professionals, such as appointment booking and professional analysis.

Subpackages:

Appointment Management:

Contains classes for scheduling and managing user appointments.

Professional Analysis:

Includes classes for advanced skincare analysis tools and features for professionals.

4. User Management:

Encompasses classes and components related to user registration, login, and account management.

5. Feedback and Improvement:

Packages related to collecting and utilizing user feedback for system improvement.

6. Reminders and Followups:

Contains classes for managing skincare reminders and followup appointments.

Dependencies:

UI depends on AI Pimple Detection System:

The user interface relies on the core functionality provided by the AI Pimple Detection System for image processing, pimple detection, and product recommendations.

AI Pimple Detection System depends on Image Processing, Pimple Detection, and Product Recommendations:

The core system is composed of subpackages, each responsible for a specific aspect of the overall functionality.

Skincare Professionals depend on User Management:

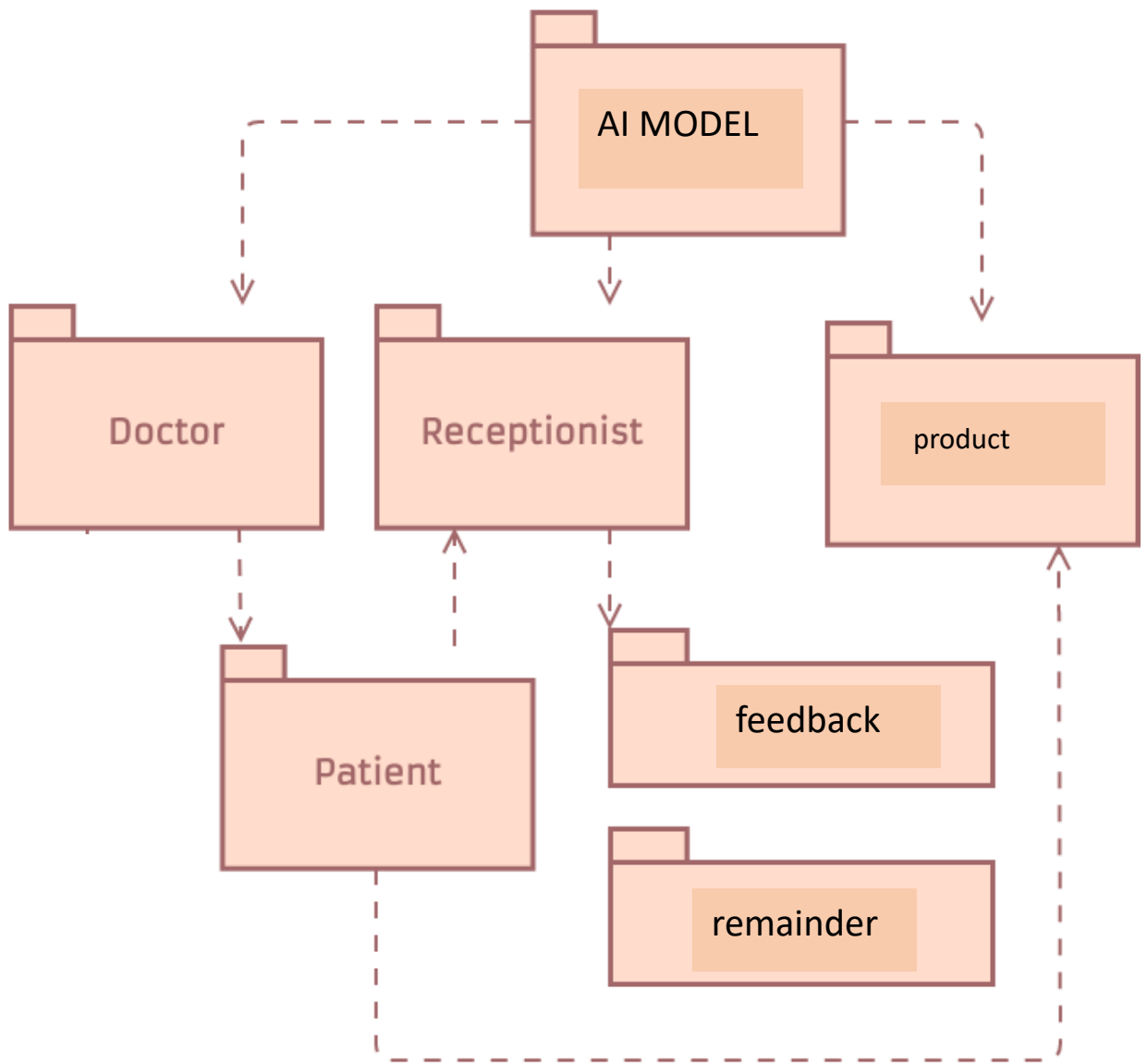
The features specific to skincare professionals, such as appointment management and professional analysis, depend on user management functionalities.

Feedback and Improvement depends on User Management:

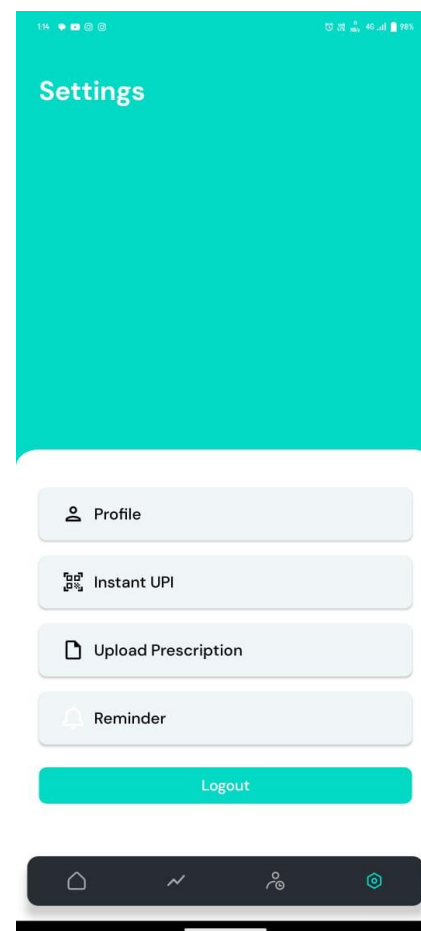
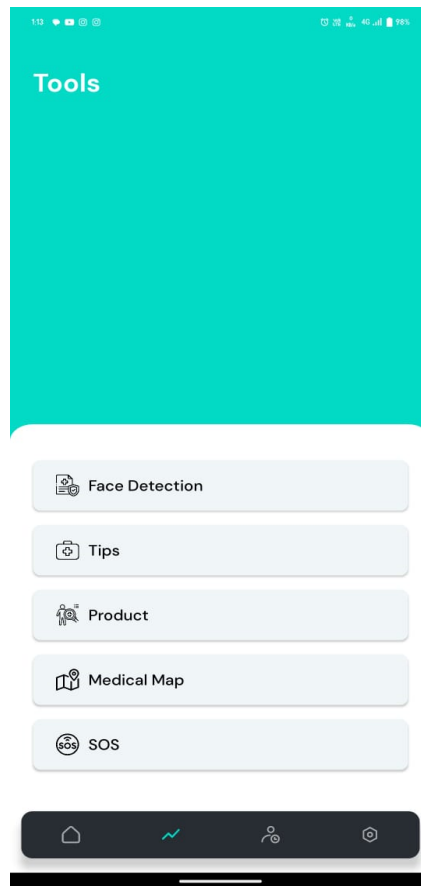
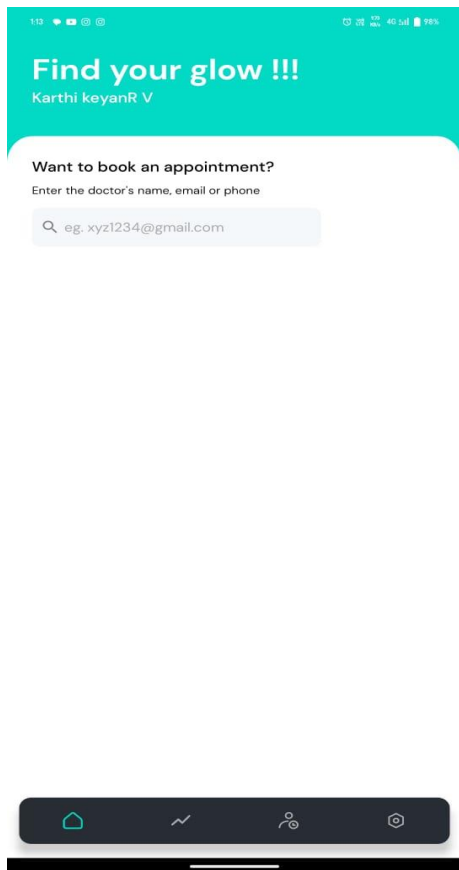
The package responsible for collecting user feedback and driving system improvement relies on user management for user identification and association.

Reminders and Followups depend on User Management:

The package managing skincare reminders and followup appointments depends on user management functionalities for user identification and association.



4.USER INTERFACE DESIGN





114 100% 40 100% 100%

Enter Your PROBLEM !

SELECT SYMPTOM

SELECT SYMPTOM

SELECT SYMPTOM

SELECT SYMPTOM


SELECT SYMPTOM

SELECT SYMPTOM

SELECT SYMPTOM

KNOW YOUR PRODUCT

NIGHT TIME ROUTINE:



Step 1: Double Cleanse Here's where it really pays to vary your routine from morning to night. Double cleansing ensures you're removing all the impurities from your skin. For maximum benefit, be sure to select products formulated for your skin type. Why you need it: Double cleansing starts with an oil-based cleanser, which is most effective for removing oil-based products like makeup and sunscreen, and ends with a water-based cleanser, which better targets sweat and bacteria. How to use it: Gently massage the oil-based cleanser into your skin for about 60 seconds, then rinse with lukewarm water. While skin is still damp, repeat the process with your water-based cleanser. Gently pat skin dry.

Step 2: Toner For your evening routine, you can use the toner you apply in the morning. Remember to choose one that's formulated for your skin type and conditions.

Step 3: Serum A nighttime serum gives your skin some extra TLC while you sleep. Choose one with AHAs or BHAs, hyaluronic acid, antioxidants or peptides. Why you need it: A serum can boost your skin's ability to repair itself at night. How to use it: Gently pat the serum into your face from the center outwards.

Step 4: Moisturizer Repeat the method you use in the morning. Depending on your skin type, you can use the same product morning and night, or you may choose one that contains SPF in the morning and a creamier, more hydrating product at night.

Step 5: Eye Cream Repeat the method you use in the morning. The same eye cream can work for both day and night, or you can select one with SPF for mornings and a heavier, more hydrating eye cream for the evening.

114 100% 40 100% 100%

you may have...

Moisturizer:

Good Things:

- Hydrates and nourishes the skin.
- Prevents dryness and flakiness.
- Helps maintain a healthy skin barrier.
- Can contain ingredients tailored to specific skin concerns (e.g., anti-aging, soothing).

Bad Things:

- Some moisturizers may be too heavy for oily skin.
- Unsuitable ingredients can cause breakouts or skin irritation.
- Inadequate application can leave the skin under-moisturized

①

5.IMPLEMENTATION AND DATABASE CONNECTIVITY

```
package com.geekymusketeers.modify.auth
import android.content.ContentValues.TAG
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.geekymusketeers.modify.databinding.ActivitySignUpFirstBinding
import io.gheok.stickyswitch.widget.StickySwitch
```

```
class SignUp_First : AppCompatActivity() {
    private lateinit var binding: ActivitySignUpFirstBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivitySignUpFirstBinding.inflate(layoutInflater)
        setContentView(binding.root)
        supportActionBar?.hide()

        binding.nextButton.setOnClickListener {
            val age = binding.ageInput.text.toString().trim()
            val isDoctor = binding.stickySwitch.getText()
```

```

if (isDoctor == "Doctor" && (Integer.parseInt(age) < 23)) {
    Toast.makeText(baseContext, "23 is the minimum age of a
    Doctor",
    Toast.LENGTH_SHORT).show()
    return@setOnClickListener
} else if (age.isEmpty()) {
    Toast.makeText(baseContext, "Enter age",
    Toast.LENGTH_SHORT).show()
    return@setOnClickListener
} else {
    val intent = Intent(this, SignUp_Activity::class.java)
    intent.putExtra("isDoctor", isDoctor)
    intent.putExtra("age", age)
    startActivity(intent)
}

}

}
}

```



```
package com.geekymusketeers.modify.auth

import android.annotation.SuppressLint
import android.content.Context
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.text.method.HideReturnsTransformationMethod
import android.text.method.PasswordTransformationMethod
import android.view.MotionEvent
import android.widget.Toast
import com.geekymusketeers.modify.HomeActivity
import com.geekymusketeers.modify.R
import com.geekymusketeers.modify.databinding.ActivitySignInBinding
import com.geekymusketeers.modify.encryptionHelper.Encryption
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ValueEventListener

class SignIn_Activity : AppCompatActivity() {
```

```
private lateinit var binding: ActivitySignInBinding
```

```
private lateinit var firebaseAuth: FirebaseAuth
```

```
@SuppressWarnings("ClickableViewAccessibility")
```

```
override fun onCreate(savedInstanceState: Bundle?) {
```

```
    super.onCreate(savedInstanceState)
```

```
    binding = ActivitySignInBinding.inflate(layoutInflater)
```

```
    setContentView(binding.root)
```

```
    initialization()
```

```
    val sharedPreferences = getSharedPreferences("UserData", Context.MODE_PRIVATE)
```

```
    val editor = sharedPreferences.edit()
```

```
// Hide and Show Password
```

```
var passwordVisible = false
```

```
binding.SignInPassword.setOnTouchListener { v, event ->
```

```
    val Right = 2
```

```
    if (event.getAction() === MotionEvent.ACTION_UP) {
```

```
        if (event.getRawX() >= binding.SignInPassword.getRight() -  
binding.SignInPassword.getCompoundDrawables().get(Right).getBounds().width()) {
```

```
            val selection: Int = binding.SignInPassword.getSelectionEnd()
```

```
            //Handles Multiple option popups
```

```
            if (passwordVisible) {
```

```
                //set drawable image here
```

```
binding.SignInPassword.setCompoundDrawablesRelativeWithIntrinsicBounds(0, 0,
R.drawable.visibility_off, 0)
```

```
    //for hide password
```

```
binding.SignInPassword.setTransformationMethod(PasswordTransformationMethod.g
etInstance())
```

```
    passwordVisible = false
```

```
    } else {
```

```
        //set drawable image here
```

```
binding.SignInPassword.setCompoundDrawablesRelativeWithIntrinsicBounds(0, 0,
R.drawable.visibility, 0)
```

```
    //for show password
```

```
binding.SignInPassword.setTransformationMethod(
```

```
    HideReturnsTransformationMethod.getInstance())
```

```
    passwordVisible = true
```

```
    }
```

```
        binding.SignInPassword.setLongClickable(false)    //Handles    Multiple
option popups
```

```
        binding.SignInPassword.setSelection(selection)
```

```
        return@setOnTouchListener true
```

```
    }
```

```
}
```

```
false
```

```
}
```

```
binding.forgotPassword.setOnClickListener {
```

```
    startActivity(Intent(this,ForgotPasswordActivity::class.java))
```

```
}
```

```
binding.toSignUp.setOnClickListener {  
    val intent = Intent(this, SignUp_First::class.java)  
    startActivity(intent)  
}
```

```
binding.loginButton.setOnClickListener {  
    val email = binding.SignInEmail.text.toString()  
    val password = binding.SignInPassword.text.toString()
```

```
if (email.isNotEmpty() && password.isNotEmpty()) {  
    if (password.length > 7) {
```

```
        firebaseAuth.signInWithEmailAndPassword(email,  
        ).addOnCompleteListener {
```

```
            if (it.isSuccessful) {  
                val u = firebaseAuth.currentUser  
                if (u?.isEmailVerified!!) {
```

```
                    val db = FirebaseDatabase.getInstance().reference  
                    val encryption = Encryption.getDefault("Key", "Salt", ByteArray(16))
```

```
                        db.child("Users").child(u.uid).addValueEventListener(object:  
ValueEventListener {
```

```
                            override fun onDataChange(snapshot: DataSnapshot) {  
                                editor.putString("uid",  
snapshot.child("uid").value.toString().trim())
```

```

        editor.putString("name",
snapshot.child("name").value.toString().trim())

        editor.putString("age",
snapshot.child("age").value.toString().trim())

        editor.putString("email",
snapshot.child("email").value.toString().trim())

        editor.putString("phone",
snapshot.child("phone").value.toString().trim())

        editor.putString("isDoctor",
snapshot.child("doctor").value.toString().trim())

editor.putString("specialist",snapshot.child("specialist").value.toString().trim())

```

```

        editor.putString("stats",
snapshot.child("stats").value.toString().trim())

        editor.putString("prescription",
snapshot.child("prescription").value.toString().trim())

        editor.putString("upi",
snapshot.child(encryption.encrypt("nulla")).value.toString().trim())

        editor.apply()

```

```

    }

    override fun onCancelled(error: DatabaseError) {
        TODO("Not yet implemented")
    }
})

```

```

startActivity(Intent(this, HomeActivity::class.java))
finish()

```

```

} else {

```

```

        u.sendEmailVerification()
        Toast.makeText(
            this,
            "Email Verification sent to your mail",
            Toast.LENGTH_LONG
        ).show()
    }
} else
    Toast.makeText(this, it.exception.toString(),
        Toast.LENGTH_SHORT).show()
    }
} else {

        Toast.makeText(this, "Password length must be greater than 8",
        Toast.LENGTH_SHORT).show()
    }

    } else {

        Toast.makeText(this, "Please enter the details!",
        Toast.LENGTH_SHORT).show()
    }
    }
}

private fun initialization() {
    supportActionBar?.hide()
    firebaseAuth = FirebaseAuth.getInstance()
}

```

```
}
```

```
package com.geekymusketeers.medify.auth
```

```
import android.content.Intent
```

```
import androidx.appcompat.app.AppCompatActivity
```

```
import android.os.Bundle
```

```
import android.widget.Toast
```

```
import com.geekymusketeers.medify.R
```

```
import com.geekymusketeers.medify.databinding.ActivityForgotPasswordBinding
```

```
import com.google.firebase.auth.FirebaseAuth
```

```
class ForgotPasswordActivity : AppCompatActivity() {
```

```
    private lateinit var binding : ActivityForgotPasswordBinding
```

```
    private lateinit var auth : FirebaseAuth
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
```

```
        binding = ActivityForgotPasswordBinding.inflate(layoutInflater)
```

```
        setContentView(binding.root)
```

```
        auth = FirebaseAuth.getInstance()
```

```
        binding.ForgotPassButton.setOnClickListener {
```

```
            val email = binding.ForgotPassEmail.text.toString().trim()
```

```
            auth.sendPasswordResetEmail(email).addOnCompleteListener {
```

```
                if (it.isSuccessful){
```

```
        Toast.makeText(this,"Email has been  
send",Toast.LENGTH_SHORT).show()  
        startActivity(Intent(this,SignIn_Activity::class.java))  
    }else  
  
        Toast.makeText(this,"Email failed to send",  
Toast.LENGTH_SHORT).show()  
    }  
}  
}
```



```
package com.geekymusketeers.modify.mainFragments

import android.annotation.SuppressLint
import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.os.Bundle
import android.os.Handler
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.view.inputmethod.EditorInfo
import android.widget.Toast
import androidx.core.view.isVisible
import androidx.fragment.app.Fragment
import com.geekymusketeers.modify.appointment.AppointmentBooking
import com.geekymusketeers.modify.RemoveCountryCode
import com.geekymusketeers.modify.databinding.FragmentHomeBinding
import com.google.firebase.FirebaseError
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.*
import com.ncorti.slidetoact.SlideToActView

class HomeFragment : Fragment() {
```

```
private var _binding: FragmentHomeBinding? = null
```

```
private val binding get() = _binding!!
```

```
private lateinit var firebaseAuth: FirebaseAuth
```

```
private lateinit var db: DatabaseReference
```

```
//Current User's data
```

```
private lateinit var userName : String
```

```
private lateinit var userEmail : String
```

```
private lateinit var userPhone : String
```

```
private lateinit var userPosition: String
```

```
private lateinit var userType: String
```

```
private lateinit var userID: String
```

```
private var userPrescription: String = "false"
```

```
//Searched doctor's data
```

```
private lateinit var searchedName : String
```

```
private lateinit var searchedEmail : String
```

```
private lateinit var searchedPhone : String
```

```
private lateinit var searchedData : String
```

```
private lateinit var searchedUID : String
```

```
private lateinit var searchedType: String
```

```
private lateinit var sharedPreferences : SharedPreferences
```

```

override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?, savedInstanceState:
Bundle?): View? {
    _binding = FragmentHomeBinding.inflate(inflater, container, false)

    firebaseAuth = FirebaseAuth.getInstance()
    val user = firebaseAuth.currentUser

    db = FirebaseDatabase.getInstance().reference
    sharedPreferences =
requireActivity().getSharedPreferences("UserData",
Context.MODE_PRIVATE)

    getDataFromSharedPreferences()

    binding.doctorData.setOnEditorActionListener { _, i, _ ->
        if (i == EditorInfo.IME_ACTION_DONE) {
            // Call your code here
            searchedData = binding.doctorData.text.toString().trim()

            if (searchedData.isNotEmpty()) {
                if (RemoveCountryCode.remove(searchedData) == userPhone ||
searchedData == userPhone || searchedData == userEmail ||
isSameName(searchedData, userName)) {
                    Toast.makeText(requireActivity(), "Stop searching yourself",
Toast.LENGTH_SHORT).show()

```

```

        binding.cardView.isVisible = false
        binding.slider.isVisible = false

    } else {
        doctorIsPresent()
    }
} else {

    Toast.makeText(requireActivity(), "Enter doctor's email /
phone", Toast.LENGTH_SHORT).show()
}
true
}
false
}

binding.slider.animDuration = 150
binding.slider.onSlideCompleteListener = object :
SlideToActView.OnSlideCompleteListener {
    override fun onSlideComplete(view: SlideToActView) {
        if (userPrescription != "false") {

```

```

        val intent = Intent(requireActivity(), AppointmentBooking::class.java)
            intent.putExtra("Duid", searchedUID)
            intent.putExtra("Dname", searchedName)
            intent.putExtra("Demail", searchedEmail)
            intent.putExtra("Dphone", searchedPhone)

            intent.putExtra("Dtype", searchedType)
            startActivity(intent)
            binding.slider.resetSlider()
        } else {
            Toast.makeText(requireActivity(), "Please upload your
prescription in settings tab", Toast.LENGTH_SHORT).show()
            binding.slider.resetSlider()
        }

        return binding.root
    }

    private fun doctorIsPresent() {

        db.child("Doctor").addValueEventListener(object : ValueEventListener
        {
            @SuppressWarnings("SetTextI18n")

```

```

override fun onDataChange(dataSnapshot: DataSnapshot) {
    for (child in dataSnapshot.children) {
        val map = child.value as HashMap<*, *>
        val sName = map["name"].toString().trim()
        val sType = map["specialist"].toString().trim()
        val sEmail = map["email"].toString().trim()
        val sPhone = map["phone"].toString().trim()
        val sUid = map["uid"].toString().trim()

        if (searchedData == sEmail || searchedData == sPhone ||
            searchedData.trim() == sName || isSameName(searchedData,sName) ||
            RemoveCountryCode.remove(searchedData) == sPhone) {
            searchedName = sName
            searchedEmail = sEmail
            searchedPhone = sPhone
            searchedUID = sUid
            searchedType = sType

            binding.textView3.isVisible = false
            binding.cardView.isVisible = true
            binding.slider.isVisible = true
            binding.doctorName.text = "Name: Dr. $sName"
        }
    }
}

```

```

        binding.doctortype.text = "Specialization: $sType"
        binding.doctorEmail.text = "Email: $sEmail"
        binding.doctorPhone.text = "Phone: $sPhone"
        return
    } else
        binding.textView3.isVisible = true
    }
}

override fun onCancelled(error: DatabaseError) {}
fun onCancelled(firebaseError: FirebaseError?) {}
})

}

private fun isSameName(searchedName: String, dbName: String):
Boolean {
    val modSearched: String = searchedName.replace(" ",
    "").toString().trim()
    val modDB: String = dbName.replace(" ", "").toString().trim()
    return modSearched == modDB;
}

override fun onStart() {

```

```
super.onStart()  
    Handler().postDelayed({
```

```
        getDataFromSharedPreference()  
    }, 1000)  
}
```

```
@SuppressWarnings("SetTextI18n", "CommitPrefEdits")
```

```
private fun getDataFromSharedPreference() {  
    userID = sharedPreferences.getString("uid","Not found").toString()  
    userName = sharedPreferences.getString("name","Not  
found").toString()  
    userEmail = sharedPreferences.getString("email","Not  
found").toString()  
    userPhone = sharedPreferences.getString("phone","Not  
found").toString()
```

```
    userPosition = sharedPreferences.getString("isDoctor", "Not  
fount").toString()
```

```
    userPrescription = sharedPreferences.getString("prescription",  
"false").toString()
```

```
    if (userPosition == "Doctor")
```

```
        binding.namePreview.text = "Dr. $userName"
```

```
    else
```



```
        binding.namePreview.text = userName
    }
}
}
```

6.CONCLUSION

In conclusion, the Pimple Detection and Cosmetic Recommendation System is designed to revolutionize personalized skincare solutions by leveraging advanced artificial intelligence and facial recognition technologies. This Software Requirement Specification (SRS) document has outlined the key features, user classes, external interfaces, and nonfunctional requirements essential for the successful development and deployment of the system. This project aims to address these challenges by developing an AI-based application, named AISkincare, that detects facial pimples through advanced image processing and provides personalized cosmetic product recommendations based on individual skin deficiencies. The key problems to be tackled include Inaccurate Pimple Detection, Limited Personalization in Product Recommendations, User-Friendly Interface, Validation and Trustworthiness, Privacy concerns. This project seeks to fill the gap in the skincare industry by providing a comprehensive, accurate, and user-friendly solution for pimple detection and cosmetic product recommendations. The nonfunctional requirements, spanning performance, security, and software quality attributes, set the standards for a robust and reliable system. From quick response times and scalability to stringent data privacy measures and accurate product recommendations, these requirements collectively contribute to the overall success of the project.